FUNDAMENTOS DE PROGRAMACIÓN

Práctica de Laboratorio: Restricciones y agregados simples

Autores: Toñi Reina, Mariano González. **Revisores: Última modificación:** 13/3/2022

Objetivos:

- Añadir restricciones a los tipos.
- Añadir criterio de igualdad a los tipos.
- Añadir criterio de orden a los tipos.
- Implementar tipos agregados (o contenedores).

1. Añade restricciones y criterios de igualdad y de ordenación

Modifica el código de los tipos definidos en el boletín L01 para que se puedan controlar las restricciones y se implementen los criterios de igualdad y de orden que se especifican para cada tipo. A continuación, prueba los tipos usando varios casos de prueba para comprobar que las implementaciones hacen lo que se espera de ellas.

2. Implementa los tipos agregados

Implementa los tipos agregados que se indican al final del documento.

Extensión de los tipos definidos en L01

1. **Cancion**

Restricciones:

- R1: el valor en segundos de la duración de una canción siempre es mayor o igual que cero.

Criterio de igualdad: dos canciones son iguales si lo son sus títulos y sus artistas.
Criterio de ordenación: las canciones se ordenan por artista y título.

2. **Pelicula**

Restricciones:

- R1: el título no puede ser null, el resto sí.
- R2: la duración debe ser mayor que cero.

Criterio de igualdad: dos películas son iguales si lo son sus títulos y su fecha de estreno.
Criterio de orden: por fecha de estreno y por título.

3. **PartidoFutbol**

Restricciones:

- R1: el nombre de los equipos no puede ser la cadena vacía.
- R2: los goles de ambos equipos no pueden tener un valor negativo.

Criterio de igualdad: dos partidos son iguales si lo son todas sus propiedades.
Criterio de orden: por fecha, si coinciden por nombre del equipo local, y si coinciden, por nombre del equipo visitante.

4. Hotel

Restricciones:
- R1: la puntuación es nula o un número mayor o igual que cero.
- R2: el número de comentarios es nulo o un número mayor o igual que cero.

Criterio de igualdad: dos hoteles son iguales si lo son todas sus propiedades.

Tipo auxiliar Coordenada
- R1: la latitud debe estar comprendida entre -90.0 y 90.0
- R2: la longitud debe estar comprendida entre -180.0 y 180.0

Criterio de igualdad: dos coordenadas son iguales si lo son todas sus propiedades.

5. Vuelo

Restricciones:
- R1: la fecha debe ser anterior al día de hoy.
- R2: el número de pasajeros debe ser menor o igual que el número de plazas.

Criterio de igualdad: dos vuelos son iguales si tienen el mismo código, origen, destino y fecha de salida.
Criterio de ordenación: los vuelos se ordenan según su código, si coinciden según origen, si coinciden según su destino, y si persiste la coincidencia, según la fecha de salida.

6. Beca

Restricciones:
- R1: el código está formado por tres letras y cuatro dígitos.
- R2: el valor de la cuantía total debe ser mayor o igual que la cuantía mínima (1500 euros).
- R3: el valor de la duración debe ser mayor o igual que la duración mínima (1 mes).

Criterio de igualdad: dos becas son iguales si lo son sus códigos.
Criterio de ordenación: las becas se ordenan por su código.

7. Persona

Restricciones:
- R1: el DNI debe estar compuesto por 9 caracteres, los ocho primeros deben ser números, y el último una letra.
- R2: la fecha de nacimiento debe ser anterior a la fecha actual.
- R3: el email debe contener el carácter '@'.

Criterio de igualdad: dos personas son iguales si lo son todas sus propiedades.
Criterio de orden: por DNI.

Definición de tipos agregados

1. Competicion

Paquetes: fp.tipos.futbol, fp.tipos.futbol.test

Propiedades:
- *nombre*, de tipo String, consultable. El nombre la competición.
- *año*, de tipo Integer, consultable. Año en el que da comienzo la competición.
- *partidos*, de tipo List<PartidoFutbol>, consultable. Es una lista de los partidos que forman parte de la competición.
- *número de partidos*, de tipo Integer, consultable. Número de partidos jugados en la competición. Se calcula a partir de la lista de partidos.

Constructores:

- C1: recibe un parámetro por cada una de las propiedades básicas del tipo, excepto los partidos, que inicialmente estarán vacíos.

Representación como cadena: una cadena con todas las propiedades del tipo, excepto los partidos, aunque sí se debe mostrar el número de partidos de la competición.

Restricciones:

- R1: el nombre y el año no pueden ser nulos.

Criterio de igualdad: dos competiciones son iguales si lo son sus nombres y sus años.

Otras operaciones:

- *void incorporaPartido(PartidoFutbol p)*: añade el partido al final de la lista de partidos, si el partido no está ya en la lista. Si está no hace nada.
- *void incorporaPartidos(List<PartidoFutbol> partidos)*: añade los partidos de la lista al final de la lista de partidos.

2. Album

Paquetes: fp.tipos.musica, fp.tipos.musica.test

Propiedades:

- *nombre*, de tipo String, consultable. El nombre del álbum
- *id*, de tipo String, consultable. Un identificador unívoco del álbum.
- *popularidad*, de tipo Integer, consultable. Un entero que representa la popularidad del álbum.
- *año de publicación*, de tipo Integer, consultable. Año de publicación del álbum.
- *tipo del álbum*, consultable. Podrá tener uno de los siguientes valores: ALBUM, SINGLE, COMPILATION.
- *canciones*, de tipo List<Cancion>, consultable. Es una lista de las canciones que forman el álbum.
- *imagenes*, de tipo List<String>, consultable. Es una lista que contendrá las rutas de las imágenes relacionadas con el álbum.

Constructores:

- C1: recibe un parámetro por cada una de las propiedades básicas del tipo, excepto las canciones y las imágenes, que estarán vacías.

Representación como cadena: una cadena con todas las propiedades, excepto las canciones y las imágenes, de las que solo se mostrará el número de elementos de las listas.

Restricciones:

- R1: el id debe tener 22 caracteres.
- R2: la popularidad debe estar comprendida entre 0 y 100, ambos inclusive.
- R3: la url de todas las imágenes del álbum debe comenzar por “http”.

Criterio de igualdad: dos álbumes son iguales si lo son sus ids.

Otras operaciones:

- *void incorporaCancion(Cancion c)*: añade la canción al final del álbum, si la canción no está ya en el álbum. Si está no hace nada.
- *void incorporaCancion(Cancion c, int posicion)*: añade la canción en la posición dada como parámetro, si la canción no está ya en el álbum. La posición debe estar en el intervalo [0, número de canciones del álbum], si no lo está, se eleva IllegalArgumentException.
- *void eliminaCancion(Cancion c)*: elimina la canción del álbum. Si la canción no está en el álbum, no hace nada.
- *void eliminaCancion(int posicion)*: elimina la canción del álbum situada en la posición dada como parámetro. La posición debe estar en el intervalo [0, número de canciones del álbum), si no lo está, se eleva IllegalArgumentException.
- *void incorporaImagen(String ruta)*: incorpora la ruta de una imagen a las imágenes del álbum.
- *void eliminaImagen(int posicion)*: elimina la imagen situada en la posición dada como parámetro. La posición debe estar en el interavalo [0, numero de imágenes), si no lo está, se eleva IllegalArgumentException.

3. ListaReproduccion

Paquetes: fp.tipos.musica, fp.tipos.musica.test

Propiedades:

- *nombre*, de tipo String, consultable y modificable.
- *canciones*, de tipo List<Cancion>, consultable.
- *número de canciones*, de tipo Integer, consultable. Se calcula a partir de la propiedad anterior.

Constructores:

- C1: recibe un parámetro para indicar el nombre de la lista. Al crearse, la lista de reproducción no contendrá canciones.

Representación como cadena: el nombre de la lista, seguido del número de canciones entre paréntesis. Por ejemplo: "Música tranquila (142 canciones)".

Criterio de igualdad: dos listas de reproducción son iguales si lo son todas sus propiedades básicas.

Otras operaciones:

- *void aleatoriza()*: cambia aleatoriamente el orden de las canciones en la lista. Utilice el método estático *shuffle* de la clase *Collections*.
- *void incorpora(Cancion c)*: añade la canción al final de la lista de reproducción.
- *void incorpora(Album a)*: añade todas las canciones del álbum al final de la lista de reproducción.
- *void incorpora(List<Cancion> canciones)*: añade todas las canciones de la lista al final de la lista de reproducción.
- *void eliminaPrimera(Cancion c)*: elimina la primera aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
- *void eliminaUltima(Cancion c)*: elimina la última aparición de la canción en la lista de reproducción. Si la canción no pertenece a la lista, se lanza IllegalArgumentException.
- *void eliminaTrozo(int primeraPosicion, int ultimaPosicion)*: elimina todas las canciones de la lista que van desde la posición primeraPosicion hasta la posición ultimaPosicion, ambas inclusive. Debe cumplirse que primeraPosicion sea mayor o igual que cero, ultimaPosicion sea menor que el número de canciones en la lista, y primeraPosicion sea menor o igual que ultimaPosicion; en caso contrario se lanza IllegalArgumentException.